CONTENTS Page

This manual describes the organization and various technical details of MSX-DOS version 2.00.

1. INTRODUCTION

This document describes various general aspects of the MSX-DOS 2.00 system which are not covered by the user documentation. Most of the topics covered are internal technical details which should not be of interest to users of the system. They are relevant for people modifying or installing the system. In particular this document contains a specification of the device driver interface which is required by OEMs who wish to install the system on their own hardware.

The following documents should be read in conjunction with this document:

MSX-DOS 2 - Command Specification
MSX-DOS 2 - Function Specification
MSX-DOS 2 - Program Interface Specification

1.1 SYSTEM REQUIREMENTS

The hardware requirements for running the MSX-DOS 2 system are as follows:

- An MSX2 compatible machine
- Must contain the mapper chip option
- Minimum of 128K of mapped RAM

The system can run without any disk controller hardware, using the built in RAM disk and disk BASIC. However normally a system will also contain a disk controller with at least one disk drive.

2. SYSTEM ORGANIZATION

The MSX-DOS 2 system is provided in two parts. The 32k disk ROM which contains the mapper support, DOS Kernel, disk BASIC and disk driver, and the system files "MSXDOS2.SYS" and "COMMAND2.COM" which provide the CP/M compatible transient program environment and the command interpreter. There are also some separate utility programs provided.

2.1 THE DISK ROM

The disk ROM can reside either in an external cartridge or inside the MSX2 machine, in any primary or expanded slot. It is divided into two 16k sections which must be in page-1 and page-2 of the same slot.

The DOS kernel is contained in the ROM in page-2 (8000h...BFFFh), but is copied at initialization time to a mapper RAM segment and actually executes in this RAM segment in page-0 (0000h...3FFFh). Whenever this code is executing, another mapper RAM segment called the kernel data segment must be present in page-2.

The other half of the ROM is in page-1 (4000h...7FFFh) and contains initialization code, disk BASIC, the RAM disk driver and (usually) a disk driver module supplied by the OEM. This code executes directly out of the ROM in page-1 when required.

There may be more than one disk ROM in the system and at initialization time each one will be called in turn. The first one to be called will become the "master disk ROM" and will be the one which runs disk BASIC and the DOS kernel. The others will become slave disk ROMs and only their disk driver modules will be used. The system can support up to four disk driver cartridges with one driver in each, and each driver can support any number of physical drives up to a total of eight. These limits include the disk driver in the master ROM if there is one. The RAM disk is included in the limit of eight drives, but does not count as taking up one of the four slots.

The system is fully compatible with MSX-DOS 1 disk drivers and so a slave cartridge can contain an MSX-DOS 1 ROM. However there are performance benefits to be gained by improving the disk drivers so this should be regarded as an interim solution. If old and new ROMs are mixed then care must be taken to ensure that the slots are arranged such that a new ROM is initialized first, since otherwise the old MSX-DOS 1 system will run.

2.2 INITIALIZATION AND BOOT PROCEDURE

As described above, the first disk ROM to be initialized will set itself up as the master ROM and will control the disk system. This ROM gets control via the "run clear" hook just before MSX BASIC starts up. What follows is a description of the initialization and boot procedure which this ROM carries out.

The fixed workspace and some other memory areas are allocated in system RAM by moving "HIMEM" downwards. This allocation includes several hundred bytes which is used for various mapper support and slot handling routines. This is necessary because the DOS kernel runs in page-0 and so when it is running the normal MSX BIOS slot handling routines are not available. The kernel code segment contains slot switching and interrupt entry points.

The two highest numbered mapper RAM segments are allocated permanently as the "kernel data segment" and the "kernel code segment", and the 16k DOS kernel is copied from page-2 of the master ROM to the kernel code segment. This code is then called in page-0 to do further initialization which includes setting up all of the data structures and variables which are required in the kernel data segment in page-2. Once this has been done, the DOS kernel is able to execute function calls.

The kernel code and data segments are then paged out and the disk BASIC environment set up. This consists of allocating various buffers in page-3 and setting up an inter-slot jump via address 0F37Dh for doing function calls. The code at 0F37Dh does not simply call the DOS kernel because it must do slot switching, paging, stack switching and also copy parameters to and from buffers in page-3. Also routines are set up to handle disk errors and program termination in an MSX-DOS 1 compatible way (using the "DISKVECT" and "BREAKVECT" vectors). See section 2.3 for more details about the disk BASIC environment and the 0F37Dh function entry point.

At this point the basic system initialization is completed and it now goes on to the boot procedure which will either start up disk BASIC or MSX-DOS. The stack is switched to a temporary stack at address C200h for the boot procedure.

The disk ROM checks to see if the "H.STKE" hook (address 0FEDAh) is still a Z-80 "RET" instruction (0C9h). If not then a cartridge must have set the hook to get control, so the ROM initializes disk BASIC and jumps to this hook. Also if there is any cartridge containing BASIC text, then disk BASIC is initialized and the BASIC program is executed.

After this the system attempts to boot up MSX-DOS. This is a two stage process. The first stage consists of trying to read a boot sector from each disk drive in turn. If one is read successfully and if the first byte is either EBh or E9h (8086 jump instructions!) then the first 100h bytes are copied to address C000h (just below the temporary stack), and location C01Eh is called with carry clear. This gives dedicated non MSX-DOS applications a chance to start up in the disk BASIC environment. Normally this code will just return.

The second stage of booting up occurs only if a boot sector was successfully read. If no boot sector was read then disk BASIC is started up. Assuming that a boot sector was read, the system selects this drive as the default and looks for a file called "MSXDOS2.SYS" on the root directory of this drive. If the file is not there then the system looks on each drive in turn for the "MSXDOS2.SYS" file. If the file is not found at all then disk BASIC is started up.

If the "MSXDOS2.SYS" file is found then it is read into page-0 starting at address 0100h and jumped to. The memory at 0000h...0100h will have been set up with all the necessary slot switching and interrupt jumps, and a null command line at address 0080h. See section 2.4 and the "Program Interface Specification" for more details of the MSX-DOS environment.

Note that unlike MSX-DOS 1, the boot sector is never called with the carry flag set and so the boot sector code is not used to actually boot up the system. Also note that the "CALL SYSTEM" command in disk BASIC can be used even if the system boots straight into disk BASIC, and it can pass a command to the "REBOOT" entry which it calls and this command will be executed by COMMAND2.COM when it is loaded and started up by MSXDOS2.SYS.

When disk BASIC is first started up, it looks for a file called "AUTOEXEC.BAS" and executes it as a BASIC program. This is only done when disk BASIC is started up automatically, not if it is entered from MSX-DOS with the "BASIC" command. In this case a command can be given to disk BASIC as a parameter to the "BASIC" command.

2.3 DISK BASIC ENVIRONMENT

The disk BASIC environment is active whenever MSXDOS2.SYS is not resident in memory. Either disk BASIC, or some other MSX application program (usually in a cartridge ROM) will be running. In this environment MSX-DOS function calls can be made by calling address 0F37Dh. A few of the functions are not available or are slightly different in this environment, the details are explained in the "Function Specification" document.

The disk ROM initialization will enter the program with the MSX BIOS ROM in page-0, the program's slot in page-1 and the RAM slot in pages 2 and 3. The RAM paging will be set up with the basic 64k of RAM paged in. This will normally be segments 0, 1, 2 and 3 but programs should NOT assume this, they can find the segment numbers by using the "GET_Pn" mapper routines. The stack will be in page-3.

The MSX system variable "HIMEM" will be set up correctly to the lowest address in page-3 RAM which is being used. A program may use memory from here on downwards in the usual way for MSX applications. Above this address will be all of the disk systems fixed work, page-3 resident code, function call and interrupt stacks and buffers for parameter copying.

The program may do any slot switching and RAM paging which it wants in pages 0, 1 and 2 but must always leave page-3 alone. If it puts anything other than the MSX BIOS ROM in page-0 then the program must of course ensure that the interrupt and slot handling entry points are present.

2.3.1 FUNCTION CALLS

When a function call to 0F37Dh is done the code there does an inter-slot call to the master ROM in page-1 (after saving register IX). The function entry code in the master ROM does a stack switch to a dedicated stack in page-3, unless this stack is already active. It then remembers the current slot selections and RAM paging state and copies any parameters into dedicated buffers in page-3. After this it enables the mapper RAM slot in pages 0 and 2, pages the kernel code and data segment it and calls the DOS kernel function dispatcher in page-0.

On return the paging and slot selections are set back to the state they were on entry, any results are copied from page-3 buffers back to the user's memory and the stack is restored. All registers including the alternate and index registers are corrupted by all function calls.

Because the inter-slot call to page-1 is done before the stack switch, the program must not have the stack in page-1 when making a function call. The stack may be in any other page and for most programs it will probably be in page-3.

The parameters which are copied are any memory resident parameters such as FCB's, fileinfo blocks and pathname strings. Because they are copied AFTER the inter-slot call has been done, they must not be in page-1 unless they are actually in the master disk ROM. Thus for example disk BASIC (which is in the master disk ROM) can use pathname strings in page-1 but any other program must copy them into some other page.

Disk transfer areas and environment strings are not copied through page-3 buffers, they are accessed using RAM paging routines. This means that they will always be accessed in RAM, in whichever RAM segment was paged into the appropriate page of the RAM slot when the function call was made. This is the case even if the RAM slot was not actually selected in that page. Thus for example if a ROM resident program (such as disk BASIC) wants to get the value of an environment string, it must copy the environment string name into RAM before making the function call, it cannot use the name string directly from ROM.

2.3.2 DISK ERRORS

When the DOS kernel detects a disk error it calls an internal error vector in page-3 at "KDSK_VECT". When the disk BASIC environment is active this vector will jump to a translation routine which translates the MSX-DOS error code into an error code compatible with MSX-DOS 1 (in fact the error code as returned by the disk driver) and then calls a user error routine via a pointer at "DISKVECT" in page-3.

This ensures compatibility of disk error routines in the disk BASIC environment while allowing more flexibility with errors in the MSX-DOS environment (see below). A program making MSX-DOS function calls from the disk BASIC environment must always have a "DISKVECT" routine defined as there is no default error handling built in to the ROM. It may be that disk BASIC will set up a default error handler which may be used by other programs.

The "DISKVECT" routine will be called with the same RAM paging and slot selections as when the function call was made, but the stack will be the internal MSX-DOS stack in page-3 not the program's stack. The error routine can make certain MSX-DOS calls but must be careful to avoid recursion. The "Function Specification" document defines which function calls are safe to make in error routines.

The parameters and results for the "DISKVECT" routine are as below. Registers A and C are compatible with MSX-DOS 1, the others are provided to allow more sophisticated error handling. It is recommended that the new error code in A' is used by all new programs since it can easily be translated into an error message using the explain function:

```
Parameters:  A  = Drive number
             C  = MSX-DOS 1 compatible error
                    (eg. not ready = 02h)
             A' = MSX-DOS 2.00 error code
                    (eg. not ready = .NRDY = FCh)
             B  = Flags - b0 set => writing
                          b1 set => ignore not recommended
                          b2 set => auto-abort error
                          b3 set => sector number is valid
             DE = Sector number (only if b3 or B is set)
Results:      C = 0 => Ignore
                  1 => Retry
                  2 => Abort
```

The "DISKVECT" routine returns a code in register C to indicate the desired response. All other main, alternate and index registers may be destroyed. If the routine returns "abort" then the "BREAKVECT" routine will be jumped to (see the next section), otherwise the error will be retried or ignored from within MSX-DOS. It is strongly recommended that a program which wants to abort a disk error should return C=2 to indicate "abort" and then carry on processing when its "BREAKVECT" routine is called. However for compatibility it is also allowable for the "DISKVECT" routine to jump back to the program without returning rather than returning a response code. In this case the program must set up its own stack before making any MSX-DOS function calls.

2.3.3 PROGRAM TERMINATION

The "BREAKVECT" vector will be called by MSX-DOS whenever a program termination condition occurs. The routine will be entered in exactly the same environment as the original MSX-DOS call was made, with the user stack active and set as if it was just about to return from the MSX-DOS call. If a termination condition occurs within a nested MSX-DOS call (presumably one made from a "DISKVECT" routine) then all nested calls will be aborted.

The "BREAKVECT" routine is passed an error code in register A which specifies the error condition which caused the termination. There is also a secondary error code in register B which will always be zero except for ".ABORT", ".INERR" and ".OUTERR" errors. These error codes are exactly the same as for a user error routine in the MSX-DOS environment as described in the "Function Specification" document under the "define abort routine" function call (although this function itself is not available under the disk BASIC environment).

2.4 MSX-DOS ENVIRONMENT

The MSX-DOS environment is set up when the "MSXDOS2.SYS" file is read in and executed by the disk ROM, and remains active until a "BASIC" command is typed to transfer back to disk BASIC. MSX-DOS can be entered either automatically at boot up time, or by a "CALL SYSTEM" command from disk BASIC. When it is started from disk BASIC, a command can be passed to be executed by the command interpreter and if no command is given then it will execute the "REBOOT.BAT" file. When it is started up automatically at boot time it will execute the "AUTOEXEC.BAT" file.

When "MSXDOS2.SYS" is loaded and executed it relocates itself to the top of available memory in page-3 and reserves some space here for buffers. It will overlay the page-3 parameter copying buffers which were used in the disk BASIC environment since it does its own parameter copying. This means that the existing 0F37Dh function entry code cannot be used, so this location is set to jump to "MSXDOS2.SYS"'s own function entry code. Another function entry point is set up at location 0005h and this is the one normally used by programs although the two are in fact identical.

"MSXDOS2.SYS" loads and runs the "COMMAND2.COM" command interpreter program. This actually runs as a normal transient program which has the capability of loading and running other programs. It patches into the jumps at 0000h and 0005h and leaves a small portion of itself resident while it runs a transient program and this code gets control when the transient program terminates.

More details about the MSX-DOS environment and other aspects of transient programs are contained in the "Program Interface Specification".

A transient program will be entered with the mapper RAM slot in all four pages and the basic 64k of RAM paged in. When making MSX-DOS function calls, the entry code in "MSXDOS2.SYS" will page the kernel code and data segments in as required but will not do any slot switching at all. This means that it is essential that the mapper RAM slot is selected in pages 0, 2 and 3 when an MSX-DOS function call is made. Page 1 will normally be the mapper RAM slot as well although it can be any other slot if required.

The function call entry code does much the same parameter copying as in the disk BASIC environment, but because there is no inter-slot call to page-1, parameters can be passed in any of the four pages. Environment strings and disk transfer areas must always be in mapper RAM, even if some other slot is in page-1, because they are accessed by RAM paging routines.

Disk errors and program terminations are handled by resident code in "MSXDOS2.SYS" which gets control from the "KDSK_VECT" and "KAB_VECT" vectors. This code has a default action of prompting the user in the case of a disk error and returning to "COMMAND2.COM" in the case of an abort. Function calls are provided to allow a transient program to define its own disk error and abort routines in place or or as well as the default ones. Details of these functions are in the "Function Specification" document.

2.5 INTERRUPTS

Because the kernel code segment is paged into page-0 when a function call is done, it must contain slot switching entry points and also an interrupt entry point. These all jump to permanently resident code in page-3 which is set up during the disk ROM initialization procedure. There are identical jumps in the kernel code segment and the page-0 TPA segment when the MSX-DOS environment is active. The interrupt entry point does a stack switch if necessary to a stack in page-3, enables the MSX BIOS ROM in page-0, calls the normal interrupt entry point in this ROM and restores the kernel code segment to page-0 afterwards.

No resetting of the RAM paging is done when an interrupt occurs. This means that when an interrupt routine is called (via the "H.TIMI" or "H.KEYI" hooks) it may have any RAM segments at all in pages 0, 1 or 2 of the RAM slot. It can of course assume that the usual page-3 RAM segment is in page-3. This is not normally a problem because the RAM slot will not necessarily be enabled in any of these pages anyway, even in the old system. If an interrupt routine wants to access RAM in any of these three pages then it must enable the RAM slot and page the required RAM page in, and it must reset both of these before returning.

3. DISK DEVICE INTERFACE

The MSX-DOS 2.00 interface to disk device drivers is compatible with the interface for MSX-DOS 1. It is designed so that MSX-DOS 2 can work with old disk drivers, and that new device drivers can be written which will make use of the improvements in MSX-DOS 2.00 and will still work with MSX-DOS 1. However a device driver can be considerably simplified if it is only required to work with MSX-DOS 2.00 and later.

A disk driver is a separate module supplied by each manufacturer to interface MSX-DOS to a particular piece of disk hardware. It is linked with the rest of the system to produce a complete disk ROM.

Up to four disk drivers can be supported, each in a separate slot, and each driver can support several physical disk drives. There is a limit of eight to the total number of drives which the system can support, including the RAM disk which provided by the master disk ROM.

All memory management required by the disk driver is purely in terms of slot switching. The disk driver need not know about the mapper chip at all. This is to ensure compatibility and also to keep the interface reasonably simple. The controlling code in the disk ROM will ensure that the correct RAM segments required for data transfer are paged in as required.

3.1 PUBLICS AND EXTERNALS

The following symbols must be declared as PUBLIC by the disk driver:

MYSIZE - Size of the page-3 RAM work area required by the driver in bytes. This should be as small as possible and does not normally include a sector buffer because there is a general purpose sector buffer always available (see "$SECBUF" below). If this is set to 0FFFFh then the driver will not be linked in. This is useful if it is desired to have a built in disk ROM with no driver.

SECLEN - Maximum sector size for media supported by this driver. For version 2.00 this must always be 512 since no other sector sizes are supported.

DEFDPB - Base address of an 18 byte "default" DPB for this driver. This is only required for compatibility with version 1, version 2.00 makes no use of the data here and so the symbol can point anywhere. See the MSX Technical Data Book for the DPB format if MSX-DOS 1 compatibility is required.

The following routines should be declared as PUBLIC by the disk driver. They may be called via an inter-slot call and so may be entered with interrupts disabled. If the routine will take a significant time to execute then interrupts should be re-enabled (without calling ENAINT).

INIHRD	- Initialize hardware
DRIVES	- Return number of drives in system
INIENV	- Initialize work area
DSKIO	- Sector read and write
DSKCHG	- Get disk change status
GETDPB	- Get disk parameter block (DPB)
CHOICE	- Return format choice string
DSKFMT	- Format a disk
MTOFF	- Turn drive motors off
OEMSTATEMENT	- Used for system expansion

A disk driver may use any of the defined addresses within the MSX system, although these must be used with care. In particular the following variables which are declared as PUBLIC in other modules of the disk ROM may be used:

$SECBUF - Address of a 512 byte temporary buffer which may be used for any purpose by the disk driver.

RAMAD0, RAMAD1, RAMAD2, RAMAD3 - Slot address of the RAM in each of the four pages. These four will always be equal in MSX-DOS 2.00.

RAWFLG - Read-after-write flag. If non-zero then each write sector operation should be verified by the driver.

DOS_VER - Version number of the DOS. Will be zero for any version earlier than 2.00. For version 2.00 it will have the value 20h. Later versions will be greater than 20h. Can be tested by multi-version drivers to optimize their operation for the current version.

The following routines are declared as PUBLIC in other modules of the disk ROM and may be used by disk drivers as required:

PROMPT - Prints a message for two drive emulation on a single drive.

DISINT - Call immediately before disabling interrupts if they will be disabled for a significant period of time.

ENAINT - Call immediately before re-enabling interrupts if DISINT was called.

SETINT - Setup routine at (HL) as a timer interrupt routine (50Hz/60Hz).

PRVINT - Calls previous timer interrupt routine, should be called at the end of the routine set up by SETINT.

GETSLOT - Get disk driver's slot address into A. Only preserves DE, IX and IY.

ENASLT - Enables a slot at address specified by A:HL.

$DOSON - Enables master disk ROM slot in page-1.
$DOSOFF - Enables RAM slot in page-1.

GETWRK - Get address of disk driver's work area into IX and HL. Only preserves DE and IY.

DIV16 - BC := BC/DE, remainder in HL.

XFER - Exactly emulates an LDIR, preserving AF, IX and IY. Before the LDIR, RAM will be put into page-1 in place of the disk driver ROM. This is used when transferring data to/from page-1.

3.2 DISK DRIVER INITIALIZATION

At initialization time the "INIHRD", "DRIVES" and "INIENV" routines of the disk driver will each be called once, it that order. None of these three routines will ever be called again.

"INIHRD" should just initialize the hardware. When this routine is called the disk driver workspace has not yet been allocated.

"DRIVES" should determine the actual number of drives connected to the controller and return this as the number of drives that it supports in register L. When this is called the disk driver workspace RAM will have been allocated. If there is only one drive connected, and the Z-flag is clear on entry, then this routine should return the value "2" and must then logically support two drives using the "PROMPT" routine. It is not acceptable to return L=0 from this function.

"INIENV" should just initialize the disk driver's work area ready for the driver to be used. The address of the work area can be determined by calling the "GETWRK" routine.

INIHRD -	Parameters:	None
	Results:	None
	May corrupt:	AF, BC, DE, HL, IX and IY.
DRIVES -	Parameters:	Z-flag. If clear (NZ) then two drive emulation should be enabled if there is only one physical drive.
	Results:	L = Number of drives supported.
	May corrupt:	F, HL, IX and IY.
INIENV -	Parameters:	None
	Results:	None
	May corrupt:	AF, BC, DE, HL, IX and IY

3.3 DISK READ AND WRITE ROUTINE

DSKIO -	Parameters:	Carry flag - clear => read set => write A = drive number 0.... B = number of sectors to transfer C = media descriptor byte DE = first logical sector number HL = transfer address
	Results:	Carry flag - clear => successful set => error If error: A = error code B = remaining sectors
	May corrupt:	AF, BC, DE, HL, IX and IY.

This routine does the actual work of reading and writing sectors to the disk. The logical sector number starts at zero and the driver must map these sectors to physical track/side/sector numbers depending on the value of the media descriptor byte passed in register C. The data must be transferred to or from RAM in the slots specified by RAMAD0...RAMAD3, using the currently selected mapper RAM segments.

In MSX-DOS 1 the transfer could be to any one of the four pages and disk drivers had to go to great lengths to do a transfer if it was in page-1. This is still necessary if a disk driver is to remain compatible with MSX-DOS 1. However MSX-DOS 2.00 will always do its transfers to page-2 or page-3 and the appropriate RAM will be directly available. Therefore if a disk driver does not need to remain compatible with MSX-DOS 1, it does not need to do any of the complex copying of code and slot switching which earlier drivers had to do. This is a considerable simplification.

Any errors should be retried a suitable number of times before being returned to the system. When an error is returned it will be reported to the user as an "Abort, Retry or Ignore" error. The defined error codes are as follows (those errors marked as "new" should only be returned for MSX-DOS 2.00 and later). "Unexpected disk change" may be returned if a "drive door opened" signal is seen, the high level code will then check to see if it is actually the same disk in the drive.

```
          0 - Write protected disk
          2 - Drive not ready
          4 - Data (CRC) error
          6 - Seek error
          7 - Record not found
         10 - Write fault (verify error)
         12 - Other error
new      18 - Not a DOS disk
new      20 - Incompatible disk
new      22 - Unformatted disk
new      24 - Unexpected disk change
```

3.4 DISK CHANGE DETECTION ROUTINE

```
DSKCHG - Parameters:   A = drive number 0....
                       B = 0
                       C = media descriptor byte
                      HL = base address of DPB
         Results:      Carry flag - clear => successful
                                    set => error
                       If success: B = 01h => not changed
                                       00h => don't know
                                       FFh => changed
                       If error:   A = error code
         May corrupt:  AF, BC, DE, HL, IX and IY.
```

This routine can be considerably simplified for disk drivers which only need to work with MSX-DOS 2.0 and later. If the disk hardware can determine whether the drive door has been opened or not then it should return B=1 (not changed) or B=FFh (changed) as appropriate. "Changed" should be returned whenever the door has been opened, without accessing the disk since it is not the driver's responsibility to decide whether it is actually still the same disk or not. If the hardware cannot determine whether the door has been opened then it should return B=0 (not sure).

This is all that is necessary for MSX-DOS 2.00. This function should never access the disk and can thus be very fast. The media byte and DPB address parameters are not required in this case.

If compatibility with MSX-DOS 1 is required then this routine must do considerably more work, involving actually working out the disk format if it returns "changed" (B=FFh) or "not sure" (B=00h). This is not documented here since it is only required for MSX-DOS 1. See the MSX Technical Data Book for details.

3.5 GET DISK PARAMETER BLOCK ROUTINE

This routine will never be called by MSX-DOS 2.00 since it does its own disk format determination by reading the boot sector. For disk drivers which do not need to be compatible with MSX-DOS 1, this routine can simply be a "RET" instruction. For drivers which do need to be compatible this function must be implemented as defined in the MSX Technical Data Book for MSX-DOS 1.

If any disk driver needs to access disks which do not have a valid parameter block on the boot sector (see section 3.7), and for which the FAT-id byte at the start of logical sector 1 cannot be interpreted as one of the standard 8 MSX floppy disk formats, then it must intercept any reads of logical sector 0 (the boot sector) and return a suitably constructed boot sector containing correct parameters for MSX-DOS to use.

3.6 DISK FORMAT ROUTINES

CHOICE - Parameters: None
Results: HL = Address of ASCIIZ choice string
May corrupt: AF, BC, DE, HL, IX and IY.

```
DSKFMT - Parameters:    A = choice from user (01h...09h)
                        D = drive number 0...
                       HL = base address of work area
                       BC = length of work area
         Results:      Carry clear => successful
                             set => error
                       If error:   A = error code
         May corrupt:  AF, BC, DE, HL, IX and IY.
```

The "CHOICE" routine will be called before formatting a disk and it simply returns a pointer to a string specifying the choices of format available. The user's response (01h to 09h) will then be passed to the "DSKFMT" routine. An extension in MSX-DOS 2.00 allows a pointer value of 0FFFFh to be returned in HL to indicate that the disk driver does not allow formatting.

The "DSKFMT" routine must format the disk according to the user's choice specified in register A. The workspace area pointed to by HL may be used for any purpose such as building up a track image if required. The disk must be formatted physically and a suitable boot sector must be written to it defining the disk format in a parameter block (see section 3.7).

If the boot sector written by the disk driver does not contain the "VOL_ID" string at the appropriate offset, then the DOS kernel will add this itself, and will put its own boot code from offset 1Eh up to a maximum of offset FFh. If the disk driver does put the "VOL_ID" string in the boot sector then it will be assumed that it has also put suitable boot code at offset 1Eh and this will not be disturbed. However in all case the DOS kernel will put a randomly generated volume id after the "VOL_ID" string and will zero the dirty disk flag.

Most disk drivers therefore do not need to put anything in the boot sector from offset 1Eh onwards. They only need to initialize this area if they want some special purpose boot code. Note that MSX-DOS 2.00 only uses the boot code on the disk for starting up disk BASIC environment programs, not for starting up MSX-DOS itself. The default boot code put on by the DOS kernel is therefore simply a "RET" instruction.

After setting up the boot sector, the driver must then initialize the FATs and root directory. The FATs should have a FATid byte in the first byte (0F8h...0FFh) and two bytes of 0FFh immediately after. The rest of the FATs and the root directory should be zeroed. This initialization is left up to the driver so that it can mark bad disk areas if desired.

The error codes which may be returned are the same as for "DSKIO" except for the following additional errors which are allowed:

12 - Bad choice parameter
14 - Insufficient memory
16 - Other error

3.7 BOOT SECTOR FORMAT

Below is the format of the boot sector for MSX-DOS 2.00 It is identical to the boot sector for version 1 except for the addition of the optional volume-id and dirty disk flag fields.

The volume-id is a 28-bit pseudo random number which the DOS kernel writes to the disk after it has been formatted. It serves as a "unique" disk identifier so that the system can tell when the disk has been changed. The dirty disk flag is used to support un-deletion and is set to zero by the DOS kernel after the disk has been formatted.

```
00h    - 0EBh  \
01h    - 0FEh   >  Dummy 8086 "JMP SHORT $" instruction
02h    - 090h  /   followed by "NOP".

03h..0Ah - "IS-SYS   "  8 character manufacturer's identity
                string, not accessed by the system.

0Bh..0Ch - Bytes per sector                 \
   0Dh   - Sectors per cluster               \
0Eh..0Fh - Reserved sectors                   |
   10h   - Number of FATs                     |   This is an
11h..12h - Number of root dir. entries        |   MS-DOS 2.00
13h..14h - Total sectors on disk              >  bios parameter
   15h   - Media descriptor                   |   block.
16h..17h - Sectors per FAT                    |
18h..19h - Sectors per track                  |
1Ah..1Bh - Number of heads                   /
1Ch..1Dh - Number of hidden sectors         /

1Eh..20h - "JP BOOT" MSX-DOS boot entry point.  Must jump
                to boot code located above this defined part
                of the boot sector.  May just be a "RET".

21h..3Fh - Reserved.

40h..45h - "VOL_ID"  string used by the system to tell
                whether the disk contains a valid volume-id
                and dirty disk flag or not.
```

46h - Dirty disk flag. Controls un-deleting on the disk. Initially zero.

47h..4Ah - 4 byte unique volume id. Each byte must be in the range 00h..7Fh selected randomly at format time by the DOS kernel.

4Bh..FFh - Available for boot code or any other machine dependent purpose. Not accessed by the system except at boot time.

100h.1FFh - Not used. This area is not copied to 0C000h at boot time, only the first 100h bytes of the sector are copied. Therefore it cannot be used for boot code.

3.8 SUMMARY OF CHANGES

This section summarizes the changes in the disk driver specification from MSX-DOS 1.

1. Sector size must always be 512 bytes.

2. MYSIZE can be 0FFFFh to disable driver.

3. DPB's not used at all. DEFDPB can point anywhere, GETDPB routine never called, DSKCHG need never update the DPB. All format determination is done by the DOS kernel reading the boot sector or the FAT-id byte.

4. Disk transfers always done to page-2 or page-3 which will always be paged in when driver is called. Disk drivers should not need to do any paging or copying of code to page-3.

5. Additional error codes defined for returning from DSKIO routine.

6. Kernel code segment will usually be in page-0 when disk driver is called. This should not matter to the disk driver at all.

7. CHOICE routine can return HL=0FFFFh to indicate that this driver does not allow formatting.

8. DSKFMT routine need not initialize root directory or FATs and need not put any boot code on the boot sector. Must put parameter block on the boot sector and may optionally put the "VOL_ID" indicator on.

++++++++++ END OF DOCUMENT ++++++++++